ARHITECTURA CALCULATOARELOR 2003/2004
CURSUL 4

1.7 Erori de comunicație

La transferarea informațiilor între diverse componente ale calculatorului sau la transmiterea de la pământ la lună şi înapoi sau, de ce nu, în cazul simplei stocări a datelor, există posibilitatea ca şirul de biți primiți, înapoi să nu fie identic cu cel original. Particulele de praf sau grăsime de pe suprafața magnetică a unui dispozitiv de stocare sau un circuit defect pot duce la înregistrarea sau citirea incorectă a datelor. Mai mult, în cazul anumitor tehnologii chiar şi radiația de fond poate altera datele stocate în memoria principală a unui calculator.

Pentru a rezolva asemene probleme au fost dezvoltate diferite tehnici de codificare care permit detectare şi chiar corectare erorilor. În prezent datorită faptului că sunt implementate pe scară largă în componentele interne ale sistemelor de calcul, ele sunt invizibile pentru cei care utilizează calculatoarele. Cu toate acestea, prezența lor este importantă şi reprezintă un aport semnificativ la cercetarea ştiințifică. De fapt, multe dintre aceste metode reprezintă metode importante ale contribuțiilor aduse de matematica teoretică. Acesta este motivul pentru care vom studia unele dintre aceste tehnici, pe care se bazează fiabilitatea sistemelor actuale.

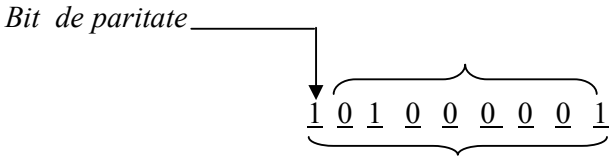
1.7.1 Biți de paritate

O metodă simplă pentru detectarea erorilor se bazează pe regula că dacă fiecare cuvânt binar manipulat are un număr impar de biți 1, atunci apariția unui cuvânt cu un număr par de biți de 1 semnalizează o eroare. Pentru a folosi această regulă, avem nevoie de un sistem în care fiecare cuvânt binar să conțină un număr impar de biți 1, cea ce se obține uşor prin adăugarea unui bit suplimentar, **bitul de paritate** (**parity bit**), la fiecare cuvânt binar dintr-un sistem deja existent (care de obicei se plasează pe poziția bitului cel mai semnificativ). (Procedând astfel codul ASCII de opt biți devine un cod de nouă biți, iar o valoare reprezentată în complement față de doi pe şaisprezece biți devine un cuvânt binar pe şaisprezece biți). În fiecare caz vom atribui noului bit valoarea 1 sau 0, astfel încât cuvântul rezultat să aibă un număr impar de1. După cum se poate observa în figura 1.27, codul ASCII pentru caracterul A se transformă în 101000001 (bitul de paritate are valoarea 1), iar codul ASCII pentru caracterul I devine 001001001 (bitul de paritate are valoarea 0). Deşi cuvântul de opt biți asociat lui A are un număr par de biți 1, iar cuvântul de opt biți asociat lui I are un număr par de biți 1, ambele cuvinte de nouă biți conțin un număr par de biți 1. După această modificare a sistemului de codificare un cuvânt binar cu nu număr par de biți 1 semnalizează faptul că s-a produs o eroare şi deci cuvântul respectiv este incorect.

Sistemul particular de paritate descris până acum poartă numele de **paritate impară** (**odd parity**), deoarece a fost astfel proiectat încât fiecare cuvânt să conțină un număr impar de biți 1. O altă tehnică utilizată **paritatea pară** (**even parity**). Într-o astfel de situație fiecare cuvânt trebuie să conțină un număr par de biți 1,iar prezența unei erori este semnalată de apariția unui cuvânt cu un număr impar de biți 1.

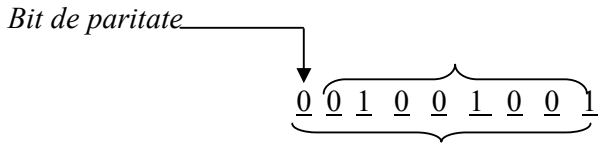
Figura 1.27 Modificarea codurilor ASCII pentru caracterele A şi I astfel încât să aibă paritate impară

Codul ASCII pentru A conţine un număr par de biţi 1



Cuvântul complet are un număr impar de biţi 1

Codul ASCII pentru I conţine un număr par de biţi 1



C*uvântul complet are un număr impar de biţi 1*

În prezent, utilizarea biţilor de paritate la memoria principală a unui calculator nu este ceva neobişnuit. Deşi ne închipuim aceste sisteme ca având celule de memorie de opt biţi, în realitate ele pot utiliza celule de nouă biţi, dintre care unul este folosit ca bit de paritate. Ori de câte ori se furnizează circuitelor de memorie un cuvânt de opt biţi pentru a fi stocat acestea îi adaugă un bit de paritate şi memorează apoi cuvântul de nouă biţi rezultat. Mai târziu, când se solicită acel cuvânt, circuitele de memorie verifică paritatea cuvântului de nouă biţi. În cazul în care controlul de paritate nu indică nici o eroare, memoria elimină bitul de paritate şi furnizează în deplină siguranţă cuvântul de opt biţi rămas. Astfel, memoria retrunează cei opt biţi de date împreună cu nu semnal de avertizare care precizează că şablonul binar rezultat s-ar putea să nu coincidă cu cuvântul stocat la început în memorie.

Şirurile lungi de biţi sunt adesea însoţite de un grup de biţi de paritate dispuşi într-un **octet de control** (**checkbyte**). Fiecare bit din acest grup este un bit de paritate asociat unui grup de biţi împrăştiaţi de-a lungul şirului de biţi. De exemplu, un bit de paritate poate fi asociat cu fiecare al optulea bit din şir începând cu primul bit din şir, în timp ce un alt bit de paritate poate fi asociat cu fiecare al optulea bit începând de la al doilea bit din şir. Procedând astfel se detectează mai uşor pachete de erori ce pot apărea în şirul de biţi iniţial, deoarece ele survin în domeniile de verificare ale mai multor biţi de paritate. Printre variantele principiului de verificare cu octet de control se numără schemele de detecţie a erorilor cunoscute sub numele de sume de control (checkbyte sums) şi controlul cu coduri ciclice (cyclic redundancy check - CRC).

1.7.2 Coduri corectoare de erori

Cu toate că utilizarea bitului de paritate permite detecţia unei erori singulare, el nu furnizează informaţia necesară pentru corectarea erori. Mulţi oameni sunt surprinşi de faptul că pot fi concepute **coduri corectoare de erori** (**error-corecting codes**), astfel încât erorile opt să fie

Figura 1.28 Exemplu de cod corector de erori

Simbol	Cod
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

nu doar detectate ci şi corectate. În definitiv, intuiția ne spune că nu putem corecta erorile dintr-un mesaj recepționat dacă nu cunoaştem deja informația conținută de mesaj. Totuşi, în figura 1.28 este prezentat un cod simplu care deține această proprietate de corecție a erorilor.

Pentru a înțelege modul în care funcționează acest cod, vom defini **distanța Hamming** dintre două cuvinte binare (denumită astfel în onoarea lui R.W. Hamming, care a început cercetarea codurilor corectoare de erori în urma dezamăgirii față de lipsa de fiabilitate a primelor maşini de calcul bazate pe relee din anii 1940) ca fiind numărul de biți prin care diferă cele două cuvinte. De exemplu, distanța Hamming dintre simbolurile A şi B din codul prezentat în figura 1.28 este patru, iar distanța dintre B şi C este trei. Caracteristica importantă a codului prezentat este aceea că oricare două cuvinte sunt separate de o distanță Hamming de cel puțin trei. Dacă în urma defectării unui dispozitiv este modificat un singur bit, eroare poate fi detectată prin faptul că rezultatul nu va fi un cuvânt de cod valid. (Trebuie să modificăm cel puțin trei biți din oricare cuvânt de cod înainte ca acesta să apară ca alt cuvânt de cod valid).

Dacă apare o singură eroare într-unul dintre cuvintele codului prezentat în figura 1.28, putem descoperi care era cuvântul de cod inițial. Într-adevăr, cuvântul modificat va fi o distanță Hamming de numai un bit față de cuvântul de cod inițial, dar la o distanță de cel puțin doi biți de orice alt cuvânt de cod valid. Pentru a decodifica un mesaj, comparăm pur şi simplu fiecare

Figura 1.29 Decodificarea cuvântului 010100 utilizându-se codul din figura 1.28

Caracter	
A	2
B	4
C	3
D	1 (Distanța cea mai mică)
E	3
F	5
G	2
H	4

şablon de biţi recepţionat cu cuvintele de cod, până găsim unul care este situat la o distanţă de cel mult un bit faţă de cuvântul verificat. Vom considera că acesta este simbolul corect pe care trebuie să-l decodificăm. Să presupunem de exemplu că recepţionăm 010100. Dacă vom compara acest cuvânt binar cu cuvintele de cod, vom obţine tabelul din figura 1.29. În consecinţă, vom putea să tragem concluzia că a fost transmis caracterul D, acesta fiind cel mai apropiat.

Aşa cum se poate observa, aplicând această tehnică asupra codului din figura 1.28 putem de fapt să detectăm apariţia a până la două erori într-un cuvânt şi să corectăm una dintre ele. Dacă am fi proiectat codul astfel încât fiecare cuvânt de cod să se afle la o distanţă Hamming de cel puţin cinci biţi faţă de celelalte cuvinte de cod, am fi putut să detectăm până la patru erori pe cuvânt şi să corectăm până la două erori. Desigur, realizarea de coduri eficiente care să aibă distanţe Hamming mari u este o tehnică simplă. De fapt, această activitate intră în preocupările unei ramurii a matematici denumită teoria decodificării algebrice.

1.7.3 Aspecte ale aplicării în practică

După cum am menţionat mai devreme, în prezent utilizarea biţilor de paritate sau a codurilor corectoare de erori este aproape întotdeauna legată de însăşi construcţia echipamentului şi poate fi arareori percepută de cel care lucrează la calculator. Utilizatorul obişnuit al calculatorului poate fi eventual interesat de aceste concepte atunci când trebuie să conecteze două echipamente, cum ar fi o imprimantă şi un calculator. În asemenea cazuri, majoritatea dispozitivelor posedă comutatoare care permit alegerea tehnicii de comunicaţie care să fie utilizată. În aceste situaţii, este responsabilitatea utilizatorului să ajusteze comutatoarele astfel încât ambele dispozitive să folosească aceiaşi tehnică de comunicare. De exemplu, vă puteţi da seama şi singur că un calculator care emite caractere cu o paritate impară şi o imprimantă care se aşteaptă să recepţioneze caractere cu paritate pară nu se vor înţelege deloc.

Decizia între a se utiliza verificarea de paritate, un cod corector de erori sau un alt sistem de tratare a erorilor depinde atât de aplicaţia corectă, cât şi de costurile acceptate pentru creşterea fiabilităţii. După cum am menţionat deja, verificarea parităţii este utilizată în multe sisteme de memorie ale calculatoarelor. Codurile corectoare de erori sunt folosite adesea de unităţile de discuri de mare capacitate, unde nevoia de corectitudine este mai mare decât costurile suplimentare apărute datorită creşterii complexităţii.

Desigur, nici un sistem de tratare a erorilor nu este complet sigur. Sistemele cu paritate nu pot detecta apariţia unui număr par de erori, iar apariţia prea multor erori într-un singur cuvânt al unui cod corector de erori poate avea ca rezultat decodificarea unui alt cuvântde cod, valid dar totuşi incorect.

Memoriile calculatoarelor pot ocazional să provoace erori, datorită variaţiilor tensiunii de alimentare, sau altor cauze. Pentru protecţia la aceste erori, unele memorii folosesc coduri detectoare sau corectoare de erori. Cand se folosesc aceste coduri, sunt adaugaţi biţi suplimentari fiecărui cuvânt de memorie, într-un mod special. Când un cuvânt este citit din memorie, biţii suplimentari sunt verificaţi pentru a sesiza prezenţa unei erori.

Pentru a întelege cum sunt tratate erorile, este necesar să privim cu atentie la ce este într-adevar o eroare. Presupuneţi că un cuvânt de memorie este compus din m biţi de date, la care

vom adăuga r biţi redundanţi, sau de control. Să presupunem că lungimea totală este n=m+r. Un şir de n biţi conţinând m biţi de date şi r biţi de control se numeşte cuvânt de cod pe n biţi.

Fiind date oricare două cuvinte de cod, să zicem 10001001 şi 10110001, se poate determina câţi biţi corespondenţi diferă. În acest caz, diferă 3 biţi. Pentru a determina câţi biţi diferă, se aplică la nivel de bit operaţia booleană XOR pentru cele două cuvinte de cod, şi se numară biţii 1 rezultaţi. Număul de poziţii în care biţi a două cuvinte diferă se numeste distanţa Hamming. Semnificaţia sa este că dacă doua coduri sunt la o distanţă d unul de altul, vor trebui considerate d erori singulare pentru a converti un cod în altul. De exemplu, codurile 11110001 şi 00110000 sunt la distanta Hamming 3, deoarece trebuie considerate trei erori singulare pentru a converti un cod în altul.

Într-un cuvant de memorie pe n biţi, toate cele 2^n combinaţii sunt permise, dar datorită modului în care se calculează biţii de control, numai 2^m din cele 2^n cuvinte de cod sunt valide. Dacă o citire a memoriei intoarce un cuvant de cod invalid, calculatorul ştie că a avut loc o eroare de memorie. Dându-se algoritmul pentru calcularea biţilor de control, se poate construi o listă cu toate cuvintele de cod permise, şi din această listă să găsim acele două cuvinte de cod pentru care distanţa Hamming este minimă. Această distanţă este distanţa Hamming a codului complet.

Proprietăţile de detecţie sau corecţie de erori ale unui cod depind de distanţa Hamming. Pentru a detecta d erori singulare, trebuie o distanţă de d+1, deoarece cu un asemenea cod nu există posibilitatea ca d erori singulare sa poată schimba un cuvânt de cod valid în alt cuvant de cod valid. În mod asemănător, pentru a corecta d erori singulare, tebuie un cod de distanţă 2d+1, deoarece astfel codurile permise sunt suficient de departe, încât şi cu d modificări codul original este mai aproape decât celălalt cod, deci poate fi unic determinat.

Ca un exemplu simplu de cod detector de eroare, consideraţi un cod cu un singur bit de paritate adăugat la biţii de date. Bitul de paritate este astfel ales încât numărul de biţi 1 în cuvântul de cod să fie par (sau impar). Un asemenea cod are distanţa 1, deoarece orice eroare singulară produce un cuvânt de cod cu paritate greşită. Cu alte cuvinte, trebuiesc două erori singulare pentru a trece de la un cuvânt de cod valid la alt cuvânt de cod valid. Se poate folosi pentru detectarea erorilor singulare. Oricând un cuvânt conţinând paritate greşită este citit din memorie, se semnalează o eroare. Programul nu poate continua, dar cel puţin nu vor fi prelucrate rezultate incorecte.

Ca un exemplu simplu de cod corector de eroare, fie codul cu numai 4 cuvinte de cod valide:

0000000000, 0000011111, 1111100000 şi 1111111111.

Acest cod are o distanţă de 5=2x2+1, ceea ce înseamnă că poate corecta erori duble. Dacă soşeste cuvântul de cod 0000000111, receptorul ştie că originalul trebuie să fi fost 0000011111 (dacă nu este mai mult de o eroare dubla). Dacă se întâmplă ca o eroare triplă să modifice 0000000000 în 0000000111, eroarea nu poate fi corectată.

Imaginaţi-vă că trebuie să proiectăm un cod cu m biţi şi r biţi de control, care va permite corectarea tuturor erorilor singulare. Fiecare din cele 2^m cuvinte de memorie permise are n cuvinte de cod nepermise la distanţa 1. Acestea sunt generate prin inversarea sistematică a fiecaruia din cei n biţi din cuvânt. Astfel, fiecare dintre cele 2^m cuvinte de memorie permise

are nevoie de modele de n+1 biţi dedicate (pentru cele n erori posibile şi modelul corect). Cum numărul total de modele de biţi este 2^n, trebuie ca:

$$(n+1)2^m \leq 2^n.$$

Folosind

$$n=m+r,$$

această cerinţa devine:

$$(m + r + 1) \leq 2^r.$$

Fiind dat m, se stabileşte o limită inferioară pentru numărul biţilor de control necesari pentru corectarea erorilor singulare. În figura 1.30 se indică numărul bitilor de control necesari pentru diferite dimensiuni ale cuvântului de memorie.

Figura 1.30 Numărul biţilor de control care pot corecta o eroare singulară.

Dimensiunea cuvântului	Biţi de control	Dimensiunea totală	Procentul depăşirii
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

Această limită inferioară teoretică se poate obţine printr-o metodă datorată lui Richard Hamming. Înainte de a analiza algoritmul lui Hamming, să privim o reprezentare grafică simplă, care ilustrează clar ideea unui cod corector de eroare pentru un cuvânt pe 4 biţi. Diagrama Venn din figura 1.31a conţine 3 cercuri, A, B şi C, care impreună formează 7 regiuni. de exemplu, să codificăm cuvantul de memorie pe 4 biţi 1100 în regiunile AB, ABC, AC şi BC, câte un bit pentru fiecare regiune (în ordine alfabetică). Această codificare este prezentată în figura 1.31.

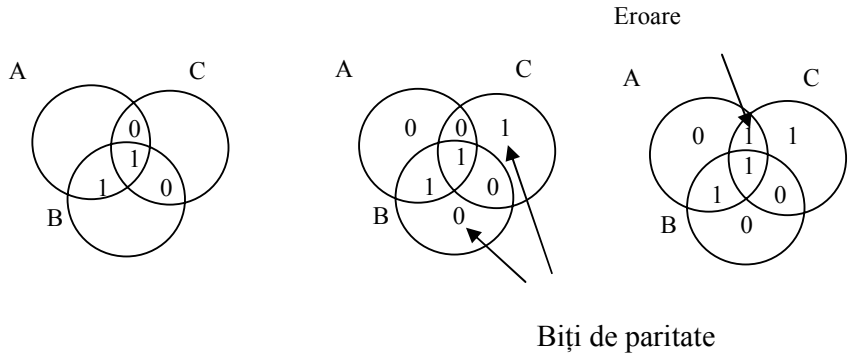


Figura 1.31 (a) Codificarea lui 1100. (b) Paritate pară adăugată. (c) Eroare în AC.

În continuare vom adauga câte un bit de paritate fiecărei regiuni goale, pentru a obține paritate pară, ca în figura 1.31b. Prin definiție, suma biților din fiecare cerc, A, B şi C, este acum un număr par. În cercul A avem cele patru numere 0, 0, 1 şi 1, a căror sumă este 2, un număr par. În cercul B avem numerele 1, 1, 0 şi 0, a căror sumă este deasemenea 2. În sfârşit, în C avem acelaşi lucru. În acest exemplu, cercurile se întâmplă să fie la fel, dar sunt posibile şi sume de 0 sau 4. Acest exemplu corespunde unui cuvânt de cod cu 4 biți de date şi 3 biți de paritate.

Să presupunem acum că bitul din regiunea AC se alterează, schimbandu-se din 0 în 1, ca în figura 1.31c. Calculatorul poate să observe acum că cercurile A şi C au paritate greşită (impară). Unica modificare a unui singur bit care corectează aceasta este să îl refacă pe AC 0. Astfel calculatorul poate corecta erorile singulare ale memoriei.

Acum să vedem cum se poate folosi algoritmul lui Hamming pentru a construi coduri corectoare de eroare pentru cuvinte de memorie de orice dimensiune. Într-un cod Hamming, r biți de paritate sunt adăugați unui cuvânt de m biți, formând un nou cuvânt de m+r biți. Numerotarea biților începe de la 1, nu de la 0, bitul 1 fiind bitul din stânga (cel mai semnificativ). Toți biții al căror număr este putere a lui 2 sunt biți de paritate, restul sunt folosiți pentru date. De exemplu, la un cuvant de date de 16 biți, se adaugă 5 biți de paritate. Biții 1, 2, 4, 8 şi 16 sunt biți de paritate, iar restul sunt biți de date. În total, cuvântul de memorie are 21 biți. În acest exemplu vom folosi (arbitrar) paritatea pară.

Fiecare bit de paritate verifică anumite poziții din cuvânt: bitul de paritate este ales astfel încât suma biților 1 din zona verificată să fie un număr par. Pozițiile biților verificate de către biții de paritate sunt:

Bitul 1 verifică biții: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.
Bitul 2 verifică biții: 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.
Bitul 4 verifică biții: 4, 5, 6, 7, 12, 13, 14, 15, 20, 21
Bitul 8 verifică biții: 8, 9, 10, 11, 12, 13, 14, 15.
Bitul 16 verifică biții: 16, 17, 18, 19, 20, 21.

În general, bitul b este verificat de biții b_1, b_2, ... b_j, astfel încât $b_1+b_2+...+b_j=b$. De exemplu, bitul 5 este verificat de biții 1 şi 4, deoarece 1+4=5. Bitul 6 este verificat de biții 2 şi 4, deoarece 2+4=6, şi tot aşa.

În figura 1.32 se arată construcția unui cod Hamming pentru cuvântul de memorie pe 16 biți 1111000010101110. Cuvântul de cod pe 21 biți este 001011100000101101110. Pentru a vedea cum funcționează corectarea erorilor, să vedem ce s-ar întâmpla dacă bitul 5 ar fi inversat printr-o variație a tensiunii de alimentare. Noul cuvânt de cod va fi 001001100000101101110, în loc de 001011100000101101110. Cei 5 biți de paritate vor fi verificați, cu următoarele rezultate:

Bitul de paritate 1 incorect (1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21 conține 5 biți 1).
Bitul de paritate 2 corect (2, 3, 6, 7, 10, 11, 14, 15, 18, 19 conține 6 biți 1).
Bitul de paritate 4 incorect (4, 5, 6, 7, 12, 13, 14, 15, 20, 21 conține 5 biți 1).
Bitul de paritate 8 corect (8, 9, 10, 11, 12, 13, 14, 15 conține 2 biți 1).
Bitul de paritate 16 corect (16, 17, 18, 19, 20, 21 conține 4 biți 1).

Numărul de biţi 1 din poziţiile 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 şi 21 ar trebui să fie par, deoarece se foloseşte paritatea pară. Bitul incorect trebuie să fie unul din cei verificaţi de bitul de paritate 1, adică 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 sau 21. Bitul de paritate 4 este incorect, deci unul din biţii 4, 5, 6, 7, 12, 13, 14, 15 sau 21 este incorect. Eroarea trebuie să fie într-unul din biţii ambelor liste, adică 5, 7, 13, 15 sau 21. Totuşi bitul 2 este corect, eliminând 7 şi 15. Similar, bitul 8 este corect, eliminând 13. În cele din urmă, bitul 16 este corect, eliminând 21. Singurul bit rămas este bitul 5, în care este singura eroare. În acest fel, erorile pot fi corectate.

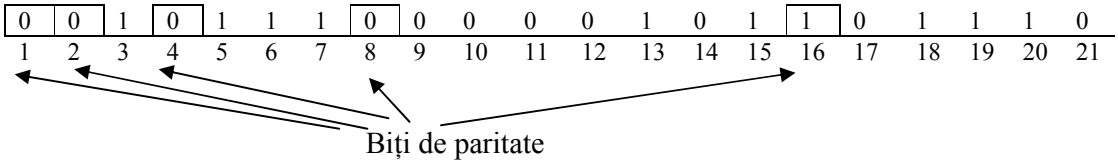


Figura 1.32 Constructia codului Hamming pentru cuvântul de memorie 111000010101110 adăugând 5 biţi de control celor 16 biţi de date

O metodă simplă de a găsi bitul incorect este de a calcula biţii de paritate. Dacă toţi sunt corecţi, atunci nu este nici o eroare (sau sunt cel puţin două). Apoi se adună toţi biţii de paritate incorecţi, numărând 1 pentru bitul 1, 2 pentru bitul 2, 4 pentru bitul 4, şi tot aşa. Suma rezultată reprezintă poziţia bitului incorect. De exemplu, dacă biţii de paritate 1 şi 4 sunt incorecţi, dar 2, 8 şi16 sunt corecţi, atunci bitul 5 (4+1) a fost modificat.

CAPITOLUL 2 : Manipularea datelor

În capitolul 1 am studiat conceptele referitoare la stocarea datelor şi memoria calculatoarelor. Pe lângă abilitatea de a stoca date, o maşină algoritmică trebuie să fie capabilă să manevreze datele conform unui algoritm. Manipularea datelor cere ca maşina să dispună de un mecanism atât pentru efectuarea de operaţii asupra datelor, cât şi pentru coordonarea secvenţelor de operaţii. În cazul calculatoarelor uzuale de astăzi acest mecanism este denumit unitate centrală de prelucrare (Central Processing Unit - CPU). Capitolul al doilea se concentrează asupra acestui dispozitiv şi a subiectelor legate de el.

2.1 Unitatea centrală de prelucrare

Într-un calculator obişnuit, circuitele care realizează diferite operaţii asupra datelor (cum ar fi adunarea sau scăderea) nu sunt conectate direct la celulele memoriei principale. De fapt, aceste circuite sunt grupate distinct într-o parte a calculatorului denumită **unitate centrală de pralucrare (Central Processing Unit - CPU**). Această unitate constă din două părţi: **unitatea aritmetico – logică** (**Arithmetic/Logic Unit**), care conţine circuitele care realizează manipularea datelor, şi **unitatea de comandă** (**Control Unit**), care conţine circuitele utilizate pentru coordonarea activităţilor maşinii.

2.1.1 Regiştri

Pentru stocarea temporară a informațiilor, unitatea centrală de prelucrare conține celule, denumite **regiştri** (**registers**), care sunt similare cu celulele memoriei principale a calculatorului. Aceşti regiştri pot fi clasificați ca fiind **regiştri de uz general** (**general - purpose registers**) şi **regiştri speciali** (**special - purpose registers**). Vom studia câțiva dintre regiştrii speciali într-un alt subcapitol. Aici, ne vom opri asupra regiştrilor de uz general.

Regiştrii de uz general servesc la stocarea temporară a datelor care sunt manipulate de CPU. Ei memorează intrările circuitelor unității aritmetico-logice şi furnizează spațiu pentru memorarea rezultatelor produse de aceasta. Pentru a putea efectua o operație cu nişte date stocate în memoria principală, unitatea de comandă trebuie să transfere datele din memorie în regiştrii de uz general, să informeze apoi unitatea aritmetico-logică în care regiştrii anume sunt stocate datele, să activeze circuitele adecvate din unitatea aritmetico-logică şi să-i indice acesteia în ce registru să depună rezultatul operației.

Este instructiv să studiem regiştrii făcând comparația cu proprietățile celorlalte elemente de memorie ale calculatorului. Regiştrii sunt folosiți pentru stocarea datelor imediat necesare pentru realizarea unei operații; memoria principală este destinată stocării datelor care vor fi necesare în scurt timp; iar dispozitivele de stocare de masă sunt utilizate pentru stocarea pe termen lung a datelor.

În multe calculatoare, la această ierarhie se adaugă un nivel suplimentar, denumit memorie cache (**cache memory**). **Memoria cache** este o memorie de mare viteză, cu timpi de răspuns similari celor ai regiştrilor unității centrale de prelucrare, situată adesea în interiorul CPU. Calculatorul stochează în această memorie o copie a acelei porțiuni din memoria pe care o utilizează în momentul respectiv. Astfel, transferurile de date care în mod normal s-ar fi făcut între regiştri şi memoria principală se fac de fapt între regiştri şi memoria cache. Modificările sunt apoi transferate în bloc în memoria principală în momente libere.

2.1.2 Interfața CPU/Memorie

Pentru transferarea cuvintelor binare între unitatea centrală a unui calculator şi memoria principală, acestea sunt conectate printr-un grup de fire denumite magistrală (bus; a se vedea figura 2.1). Prin intermediul magistralei, unitatea centrală de prelucrare poate să extragă (să citească) date din memoria principală, furnizând adresa celulei de memorie dorite, împreună cu un semnal de citire. Similar, CPU poate plasa (scrie) date în memorie indicând adresa celulei de memorie destinație şi datele care trebuie stocate, împreună cu un semnal de scriere.

Figura 2.1 Arhitectură unitate centrală de prelucrare/memorie principală

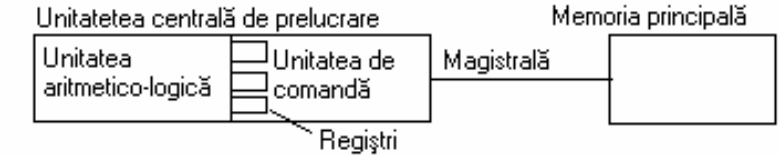


Figura 2.2 Adunarea unor valori stocate în memorie

Pasul 1: Se citeşte din memorie una dintre valorile care trebuie adunate şi se plasează într-un registru.
Pasul 2: Se citeşte din memorie cealaltă valoare care trebuie adunată şi se plasează într-un alt registru.
Pasul 3: Se activează circuitul de adunare având ca intrări regiştrii utilizaţi la paşii 1 şi 2.
Pasul 4: Se stochează rezultatul în memorie.
Pasul 5: Stop

Dacă analizăm acest mecanism, constatăm că efectuarea unei operaţii cum ar fi adunarea datelor stocate în memoria principală înseamnă mai mult decât execuţia unei simple operaţii de adunare. Procesul presupune atât implicarea unităţii de comandă, care coordonează transferul informaţiilor către regiştri şi memoria principală, cât şi a unităţii aritmetico-logice, care efectuează operaţia de adunare atunci când unitatea de comandă îi cere acest lucru.În figura 2.2 este prezentat în detaliu procesul adunării a două valori stocate în memorie.